COBB

INSIDE VISUAL dBASE™

Tips & techniques for Visual dBASE and dBASE for Windows

August 1996 • Vol. 3 No. 8
US $8.50

Working with RGB colors

Let's be honest—dBASE's color codes are weird. They've never corresponded to any other product's color codes, and their construction is just plain odd. Would you have thought that RB+ should be the designation for magenta or that N makes good sense for black? Probably not. In any case, if you've worked with dBASE for DOS for any length of time, you've adjusted to the crazy scheme shown in Table A.

Now, along comes Windows. For backwards compatibility, Visual dBASE recognizes and uses the dBASE for DOS color codes. Windows, however, uses a completely different technique for specifying colors. It's arguable whether the Windows conventions are any easier to understand than the old dBASE color codes, but Windows does allow for a much larger spectrum of colors.

You need to understand how to use colors in your applications, given the wondrous array of colors and resolutions available on modern video cards. As a Windows programming and development environment, Visual dBASE also recognizes Windows color identifiers, which are combinations of red, green, and blue (RGB) intensity values. In this article, we'll explore how you can make the best use of color in your applicatons.

Table A: *dBASE for DOS color codes*

Color	Code
Black	N
Blue	B
Green	G
Cyan	GB
Red	R
Magenta	RB
Brown	RG
White	W
Blank	X
Bold	+
Blink	*

IN THIS ISSUE

Understanding RGB

An RGB color value is a simple integer value that you can break into three parts. Each part specifies an intensity level between 0 and 255 for the colors red, green, and blue.

You'll sometimes see RGB values listed as three separate values. For instance, Visual dBASE's GetColor( ) function returns a string of three numbers separated by commas. RGB values are also frequently represented as six-digit hexadecimal numbers, ostensibly because they're more natural and easier to decipher. (You can decide that for yourself.)

Listing A on the next page is from the [COLORS] section of my WIN.INI file. I set my background color to black. The Background entry in this section lists the three values as 0, 0, and 0—that is, no intensity for any of the three colors.

Now, compare the Background entry to the Menu entry. I like my menus white. Specifying all values at maximum intensity produces white, so the Menu entry lists 255, 255, and 255 as the values for red, green, and blue.

Between these two extremes lie entries like ActiveTitle. The ActiveTitle entries are 0 for red, 64 for green, and 128 for blue. With these settings, my title bars are blue, with a touch of green added to produce a subtle shading difference from pure blue. You can produce various levels of gray by specifying RGB values of equal value, usually greater than 0 (pure black) and less than 255 (pure white).

Most of Visual dBASE's controls have a ColorNormal property. Some also have a

ColorHighLight property. You specify these properties as strings in two parts, separated by a slash to indicate foreground/background color combinations. If you omit the slash and second value, Visual dBASE assigns only the foreground color.

You can assign three different kinds of strings to the Color properties. First, the old dBASE color code values like "G/B" or "BR+/W" work just fine. In addition, Visual dBASE recognizes current system values. These special string values are shown in Table B.

Listing A: *The [COLORS] section of WIN.INI*

```
[colors]
Background=0 0 0
AppWorkspace=0 128 64
Window=255 255 255
WindowText=0 0 0
Menu=255 255 255
MenuText=0 0 0
ActiveTitle=0 64 128
InactiveTitle=192 192 192
TitleText=255 255 255
ActiveBorder=192 192 192
InactiveBorder=192 192 192
WindowFrame=0 0 0
Scrollbar=192 192 192
ButtonFace=192 192 192
ButtonShadow=128 128 128
ButtonText=0 0 0
GrayText=0 0 0
Hilight=0 128 128
HilightText=0 0 0
InactiveTitleText=0 0 0
ButtonHilight=255 255 255
```

Table B: *Current system values recognized as color codes*

ActiveBorder	Scrollbar
BtnText	BtnFace
InactiveCaptionText	HighlightText
ActiveCaption	Window
CaptionText	BtnHighlight
Menu	InactiveBorder
AppWorkspace	WindowFrame
GrayText	BtnShadow
MenuText	InactiveCaption
Background	WindowText
Highlight	

The third type of string is an eight-character expression that begins with 0x and ends with a six-digit hexadecimal RGB value. Each of the three pairs of hex digits that make up this last portion must be a value between 00 and FF. Any oddball characters or spaces will produce an error message. Note that these types of color designations are strings. Properties such as ColorNormal and ColorHighlight won't recognize non-character arguments.

Coloring by function call

While dBASE allows you to use the current system colors, it doesn't provide a way for you to set them from within your dBASE applications. However, setting the system colors is a fairly simple task, and Visual dBASE's WINAPI.H file provides a number of #defines and a couple of EXTERNed functions you can use for just that purpose. However, those functions don't recognize dBASE's proprietary color scheme or the special system color names. Instead, you must use RGB values.

The two API functions you'll use to get and set system colors are GetSysColor( ) and SetSysColors( ), respectively. Note that GetSysColor( ) is very straightforward: It returns a number (the RGB value) and takes as its single argument an index number indicating the system element whose color you want. WINAPI.H contains several #define statements to provide the appropriate index number. We pruned the program RGBCOLOR.PRG, shown in Listing B, from WINAPI.H. You can #include this code or even run it from your own source files, as you'll see shortly.

As we've mentioned, dBASE's GetColor( ) function returns an RGB value as a string of numbers separated by commas. You need to convert this value any time you use it to call GetSysColor( ) or SetSysColors( ). Listing C contains three routines you can use to accomplish the various conversions. (Some of the details of these functions are obscure, to say the least, and we won't take time to discuss them in depth here. Consult a Windows API manual or help file if you're interested in the nitty-gritty, especially in the SetSysColors( ) function call.)

Using these functions, you can change the system color of virtually any interface element. However, these changes aren't recorded permanently in the file WIN.INI—you must perform that additional step if you want to implement the changes in the same way you would by invoking the Control Panel.

For example, the code in Listing C will change the color of the active title bar to the color you specify from the color picker. Once you've typed in the RGB2Str function

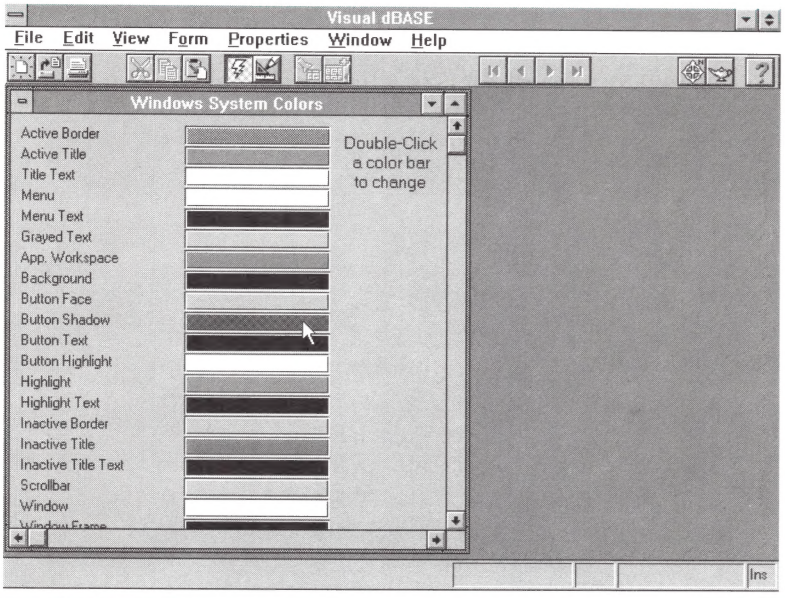
Listing B: *Helpful RGB color details stolen from WINAPI.H*

```
* RGBCOLOR.PRG
* The following EXTERNS and
* #defines are taken from WINAPI.H
*
EXTERN CLONG GetSysColor(CINT)  USER
EXTERN CVOID SetSysColors(CINT,CSTRING,CSTRING) user

********** GetSysColor() display elements

#define COLOR_ACTIVEBORDER        10
#define COLOR_ACTIVECAPTION        2
#define COLOR_APPWORKSPACE        12
#define COLOR_BACKGROUND           1
#define COLOR_BTNFACE             15
#define COLOR_BTNSHADOW           16
#define COLOR_BTNTEXT             18
#define COLOR_CAPTIONTEXT          9
#define COLOR_GRAYTEXT            17
#define COLOR_HIGHLIGHT           13
#define COLOR_HIGHLIGHTTEXT       14
#define COLOR_INACTIVEBORDER      11
#define COLOR_INACTIVECAPTION      3
#define COLOR_MENU                 4
#define COLOR_MENUTEXT             7
#define COLOR_SCROLLBAR            0
#define COLOR_WINDOW               5
#define COLOR_WINDOWFRAME          6
#define COLOR_WINDOWTEXT           8
#define COLOR_INACTIVECAPTIONTEXT 19
#define COLOR_BTNHIGHLIGHT        20
```

Listing C: *Conversion routines for working with RGB values*

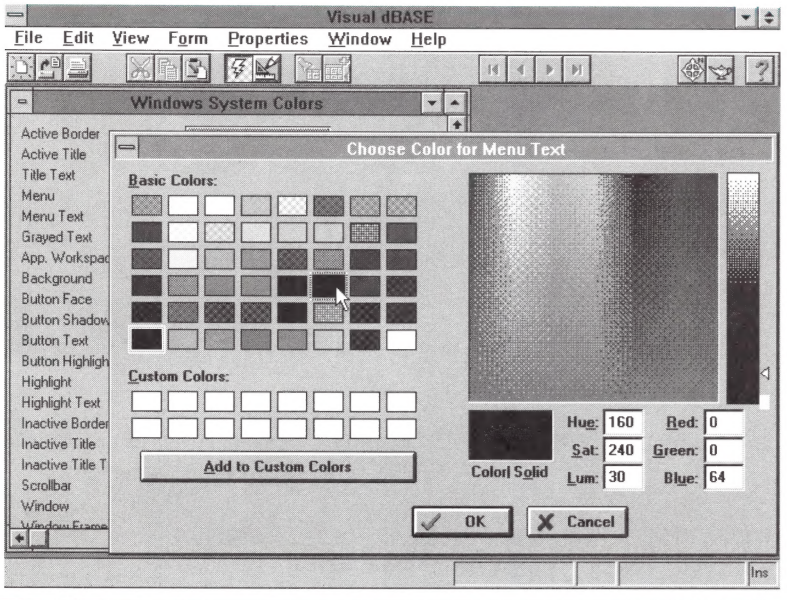
```
* INDEXSTR.PRG: returns a string appropriate for
    SetSysColors() API function
*
procedure indexstr
parameter nIndex
return CHR(bitand(nIndex,255))+CHR(bitrshift(nIndex,8))

* RGB2NUM.PRG: Converts GetColor's "Red,Green,Blue" return
*              value string to a numeric value
*
procedure rgb2num
   parameters cRGB
   local r,g,b,n1,n2
   n1 = at(",",cRGB)         && Find the commas
   n2 = at(",",cRGB,2)
   if (n1 = 0) .or. (n2 = 0) && Are they both there?
      msgbox("Invalid RGB combination specified!",;
            "Error in RGB parameter")
      return -1
   endif
   r = val(left(cRGB,n1-1))  && Convert each to number
   g = val(substr(cRGB,n1+1,n2-n1-1))
   b = val(substr(cRGB,n2+1))

return(bitlshift(r,16)+bitlshift(g,8)+b)

*
* RGB2STR.PRG: Converts GetColor's "Red,Green,Blue" return
*            value string to a string value for SetSysColor
*
procedure rgb2str
   parameters cRGB
   local r,g,b,n1,n2
   n1 = at(",",cRGB)         && Find the commas
   n2 = at(",",cRGB,2)
   if (n1 = 0) .or. (n2 = 0) && Are they both there?
      msgbox("Invalid RGB combination specified!",;
            "Error in RGB parameter")
      return -1
   endif
   r = val(left(cRGB,n1-1))  && Convert each to number
   g = val(substr(cRGB,n1+1,n2-n1-1))
   b = val(substr(cRGB,n2+1))
return chr(r)+chr(g)+chr(b)
```

Figure A



The WINCOLOR form lets you reset system colors interactively.

Figure B



When you invoke the GetColor( ) function, it opens this dialog box, where you can pick a new color for a system value.

from Listing C, you can change the color of the active title bar to the color you specify from the color picker. Just type the following into the Command window:

```
nTitleBar = 2
EXTERN CVOID
➥ SetSysColors(CINT,CSTRING,CSTRING)
➥ user
SetSysColors(1,indexstr(nTitleBar),
➥ rgb2str(getcolor()))
```

Now we'll take what we've covered so far and create a form that allows the user to change the system colors interactively. In Listing D, you'll find the code for WINCOLOR.WFM, which uses the concepts and functions we've presented to produce the interactive form shown in Figure A. WINCOLOR.WFM displays a small rectangle with the current color for each of the system values, as obtained from the GetSysColor( ) API call. A double-click on the rectangle invokes the GetColor( ) function, which displays the dialog box shown in Figure B, and changes the color using a SetSysColors( ) call.

WINCOLOR.WFM does all its dirty work in the Setup procedure. Rather than create all the text and rectangle controls interactively, I wrote a FOR loop that creates the controls dynamically. A literal array provides a simple way to set up the strings and numeric values used for each system color entry. You can modify or extend this listing as needed, should you wish to add color entries not already in the list.

Wrapping up

You can make your Visual dBASE applications more fun to use by creating visually exciting color combinations. Using the techniques and code presented in this article, you can produce wildly snazzy forms. You can even ensure that your forms follow the user's preferences exactly by checking the current system color values and adjusting the ColorNormal and ColorHighlight properties accordingly. ❖

Listing D: *WINCOLOR.WFM*

```
** END HEADER * do not remove this line*
* Generated on 03/28/96
*
parameter bModal
local f
f = new WINCOLORFORM()
if (bModal)
   f.mdi = .F. && ensure not MDI
   f.ReadModal()
else
   f.Open()
endif
CLASS WINCOLORFORM OF FORM
   this.OnOpen    = CLASS::SETUP
   this.Text      = "Windows System Colors"
   this.Height    = 20
   this.Width     = 60
   this.ScrollBar = 1
   this.Left      = 1.5
   this.Top       = 1

   DEFINE TEXT HelpText OF THIS;
       PROPERTY;
         Alignment 10,;
         Text "Double-Click a color bar to change",;
         Height 3.8818,;
         FontSize 10,;
         Width 12.666,;
         FontBold .F.,;
         Left 46,;
         Top 0.8232

   Procedure Setup
      PRIVATE cCode, cObj
      EXTERN CLONG GetSysColor(CINT)  user
      EXTERN CVOID SetSysColors(CINT,CSTRING,CSTRING) user

      form.aColors = {"Active Border"        ,10,;
                      "Active Title"         , 2,;
                      "Title Text"           , 9,;
                      "Menu"                 , 4,;
                      "Menu Text"            , 7,;
                      "Grayed Text"          ,17,;
                      "App. Workspace"       ,12,;
                      "Background"           , 1,;
                      "Button Face"          ,15,;
                      "Button Shadow"        ,16,;
                      "Button Text"          ,18,;
                      "Button Highlight"     ,20,;
                      "Highlight"            ,13,;
                      "Highlight Text"       ,14,;
                      "Inactive Border"      ,11,;
                      "Inactive Title"       , 3,;
                      "Inactive Title Text"  ,19,;
                      "Scrollbar"            , 0,;
                      "Window"               , 5,;
                      "Window Frame"         , 6,;
                      "Window Text"          , 8}

      FOR i = 1 to form.aColors.size step 2
         cObj = "text"+ltrim(str(i))
         DEFINE TEXT &cObj OF form property ;
            top      (i/2),;
            left     1.5,;
            Height   0.95,;
            width    20,;
            fontbold .f.,;
            text     form.aColors[i]

      cObj = "rectangle"+ltrim(str(i))
      cCode = "{; form.ChangeColor("   + ;
              str(form.aColors[i+1])   + ;
              ",'Choose Color for "+form.aColors[i]+"'" + ;
              "); this.ColorNormal = '0x'+itoh(GetSysColor(" + ;
                  str(form.aColors[i+1]) + "))}"

       DEFINE RECTANGLE &cObj OF form property ;
          OnLeftDblClick &cCode,;
          Text        "",;
          Height      0.95,;
          Left        24,;
          Width       20,;
          Top         (i/2),;
          colornormal "0x"+itoh(GetSysColor(form.aColors[i+1]))
      NEXT

   Procedure ChangeColor(nIndex,cTitle)
      local cColor
      cColor = GetColor(cTitle)
      if .not. isblank(cColor)
         SetSysColors(1,indexstr(nIndex),rgb2str(cColor))
      endif

ENDCLASS
```

Customizing the desktop's typeface

When you're working from the Command window, you're accustomed to seeing Visual dBASE always display your commands and the output in the same typeface and font. For instance, Figure A shows the default MS San Serif Bold 8-point typeface in the input pane and the Terminal 9-point typeface in the results pane.

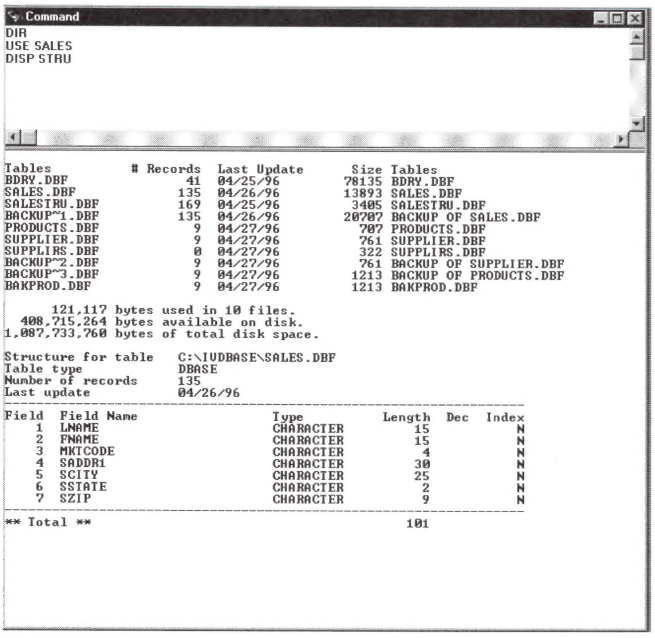
You may have wondered how you can change the typeface Visual dBASE uses in the Command and Results windows. Some people might prefer a larger, bolder look for the text, to make the commands and the results easier to read onscreen. In this article, we'll show you how to customize your Visual dBASE environment by changing the style or size of the font the program uses to display information.

Command window properties

To choose a new font for your Visual dBASE Command window, open the Properties menu and choose the Command Window Properties... option. When the dialog box shown in Figure B appears, click on the Tool icon ( ) next to the Input Pane label.

When the Font dialog box appears, choose a new font size or style and click OK. While you're experimenting, you can preview the new font in the Sample section. Figure C shows our Command window after we selected the MS Serif 14-point regular font for the input pane and Courier New 12 for the results pane.

Figure A



By default, Visual dBASE uses 8-point type for the Command window and 9-point type for the Results window.

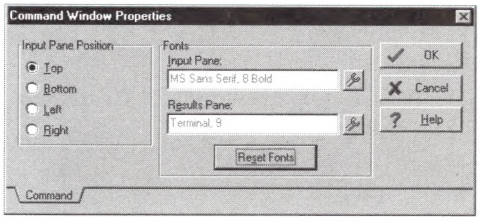
Restoring the defaults

When you change a font associated with the Command window's input and results panes, Visual dBASE saves your changes when you exit the program. To restore the default fonts, simply choose the Command Window Properties... option from the Properties menu. Then, when the Font dialog box appears, click the Reset Fonts button.

About the desktop...

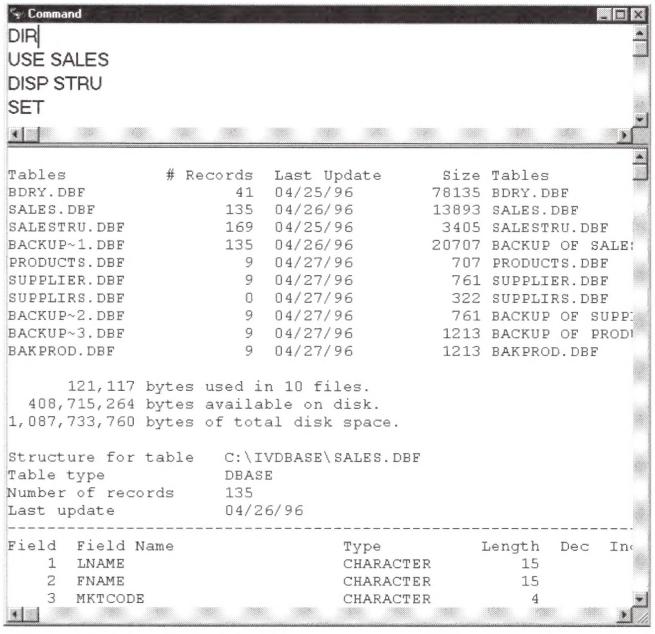
As you may have noticed, an option named Desktop Properties... appears under the Properties menu. Use this option to set various defaults for the Visual dBASE environment. Note that only one option affects the way text appears onscreen. Click the Application tab, then click the Tool icon for the Status Bar Message Font option. Choosing a new font in this way affects only messages appearing in the Status Bar at the bottom of the Visual dBASE desktop. ❖

Figure B



You'll use this dialog box to customize the appearance of your Command window.

Figure C



You can make the Command window's text easier to read by changing the size or style of the typeface.

Borland's technical support options

Borland provides several free and for-a-fee technical support options. In this article, we'll briefly summarize those options for you.

Information by FAX

TechFAX is a free service that allows you to obtain Technical Information documents—also known as TIs—via your FAX machine. To receive a categorized list of TIs, call (800) 822-4269 and request Document #2624.

Contract assistance

Borland offers two types of contract technical support. The Assist program, which is based on a yearly contract, allows you to make an unlimited number of calls. However, Assist is geared toward usability issues—the technical support personnel won't work with your code during these calls.

The Advisor program, on the other hand, covers any topic from usability to programming. Advisor is also a paid line, but you pay as you go.

The Expert Assist program is the most popular among users. In fact, it's fairly common for a customer to purchase an Expert Assist plan, then also use the Advisor line when he or she needs help writing or debugging code. For information about contract support, contact Borland directly at (800) 841-8180.

Online with CompuServe

If you subscribe to CompuServe, you can obtain dBASE support via GO dBASE. The Visual dBASE forum offers threaded discussion for dBASE developers and users. (If you have Internet access but aren't a member of CompuServe, visit **www.compuserve.com** to find out how to activate this service.)

The dBASE forum discussion topics include installation, object-oriented programming, the Visual dBASE compiler, Customer Service, and general news about Borland and its products. The forum libraries contain product patches, utilities, FAQs (Frequently Asked Questions), and more TIs.

Online newsgroup support

For Visual dBASE support on the Internet, try posting a question on the Xbase newsgroup at the following address: **news:comp.databases.xbase.misc**. Many eyes will see your question, so you're likely to receive multiple responses. The newsgroup is frequented by new and expert dBASE users, Borland employees—including Visual dBASE developers and technical support members—and Borland TeamB members. Although most areas of discussion on this newsgroup pertain to Visual dBASE, it's an open forum for all Xbase-related products, including dBASE for DOS.

Information via BBS

To obtain technical information via your modem, call (408) 431-5096 and have your modem set to 8-N-1. When you connect, join the dBASE Conference. The BBS offers files, TIs, and utilities.

Reporting problems

If you experience a problem with Visual dBASE, the ideal method for reporting it is with the Customer Problem Report (CPR). This form is available for downloading or viewing as TI-8810 on the dBASE Web Site, from the dBASE forum on CompuServe, or via fax at (800) 822-4269. Once you've completed the report, you can fax it directly to Borland's technical support department for further evaluation.

When you report a possible problem, Borland's ability to reproduce that problem is the key to finding a solution. Before reporting, narrow down the possible sources of the problem to the least amount of code. In addition, try to document the exact steps that produced the problem. Don't forget to include statistics about your system configuration.

Occasionally, you may find that you can't reproduce the problem outside the context of your application, or you may not have the expertise to narrow it down. If you have difficulty, TeamB on CompuServe will be glad to help out. Or, go ahead and submit the CPR, and a Borland Tech Support Engineer will assist you. ❖

Another approach to creating your own Find Records dialog box

*In the May 1996 issue's "*Adding a Find Records Dialog Box to Your Applications,*" we showed you how to create a custom Find Records dialog box. This month, we present another approach to creating a custom Find feature. This technique was submitted by Paul Mahar, a test engineer at Borland. Paul is the author of* Visual dBASE 5.5 Unleashed *(Borland Press/SAMS Publishing, ISBN: 0-672-30877-0). For more information, visit Paul's home page at http://ourworld.compuserve.com/homepages/mahar/.*

Visual dBASE provides a powerful dialog box for finding records. Figure A shows how to search for "Shook, Travis" in the ARTIST field. You can access the Find Records dialog box from the default form menu, the table records menu, and the Command window menu.

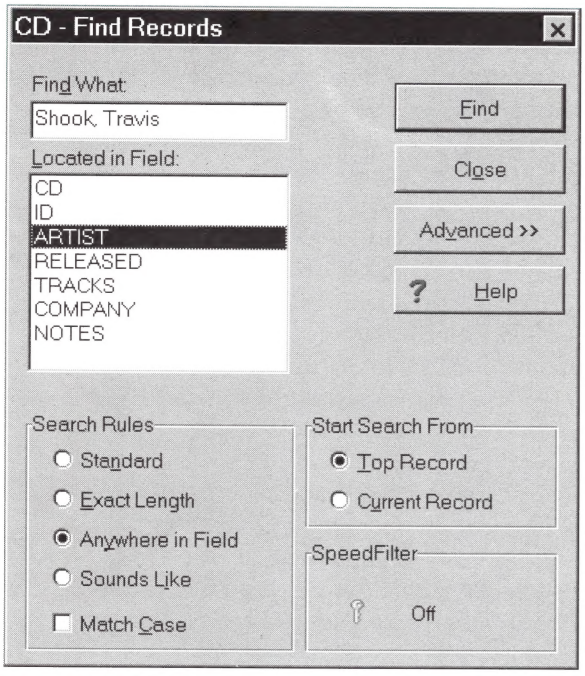
As you develop your own applications, you might want to give your end users similar functionality. Unfortunately, there's no method to open the native Find Records dialog box from a custom menu. In this article, we'll show you how to create a form that mimics some of the behavior of the Find Records dialog box.

To start, you must decide what features of the Find dialog box are appropriate for your users. For most applications, you can simplify the dialog box greatly by eliminating all the options that appear when you click the Advanced button.

Start with the least specific search rules, which are the most likely to find a matching record. Those rules allow the search string to appear anywhere in the field and are case insensitive. This behavior is similar to the default Find options in a word processor such as Microsoft Word.

The only required controls in the Find dialog box are the entry field for the value to be found, the list box for selecting the field, the Find button, and the Close button. All the other controls are optional. For the example presented in this article, the only other controls you'll add are the static labels for the entry field and the list box.

Figure A



Using the Visual dBASE Find Records dialog box is easy, but the dialog box isn't available from a custom menu.

Designing the form

You can quickly lay out your dialog box in the Form Designer. To do so, open the Form Designer and drag and drop two buttons onto the form. Use the Inspector to set the first button's Text property to *&Find* and the second button's Text property to *Cl&ose*. (The ampersands indicate hot keys—they don't appear on the buttons.) While you're changing the text properties, change the Name properties to *PUSHFIND* and *PUSHCLOSE*. You can find the Name property in the Identification properties group.

As a general rule, you can't find something before you know what to look for. To enforce this rule in your dialog box, set the enabled property of PUSHFIND to False (.F.). Later, you'll add an event to enable the button.

Next, add an entry field, a list box, and two text controls. Then, move the text controls so they appear as labels for the entry field and list box. Use the Inspector to set the Text property of the first label to *Fin&d What:*. Set the second label to *&Located in Field:*.

At this point, you've finished adding controls. Now it's time to configure the form. Set the form's MDI property to false, and resize the form so it looks similar to the native Find Records dialog box. Figure B shows the form after we added all the controls. It's a good idea to save the form before continuing.

The native Find Records dialog box shows the current table name, along with *Find Records* in the caption. Your dialog box can display a similar caption. Just add a simple code block to the form's OnOpen event in the form

```
{;FORM.text = ALIAS()+" - Find Records"}
```

This code block uses the ALIAS( ) function to get the current table name and place it in the caption along with the name of the dialog box.

Filling the list box

You can use the DataSource's STRUCTURE option to fill the List Box with all the fields from the current table. However, if you do so, your search must work with every field and every field type. Some field types—such as binary and OLE fields—aren't applicable to searches.

To fill the list box with only the appropriate field types, you can create an array containing the desired fields and use it with the DataSource property. The list box's OnOpen event is a good place to define this process. Change the name of the list box to ListFields and create an OnOpen event for it. Enter the procedure shown in Listing A for the OnOpen event.

For this dialog box, the allowable field types are character (C) and memo (M). The FOR...NEXT loop goes through all the fields in the current table and adds to an array only those fields of character or memo type. The array becomes the DataSource for the list box. As an added convenience, the procedure keys in a down arrow to select the first field.

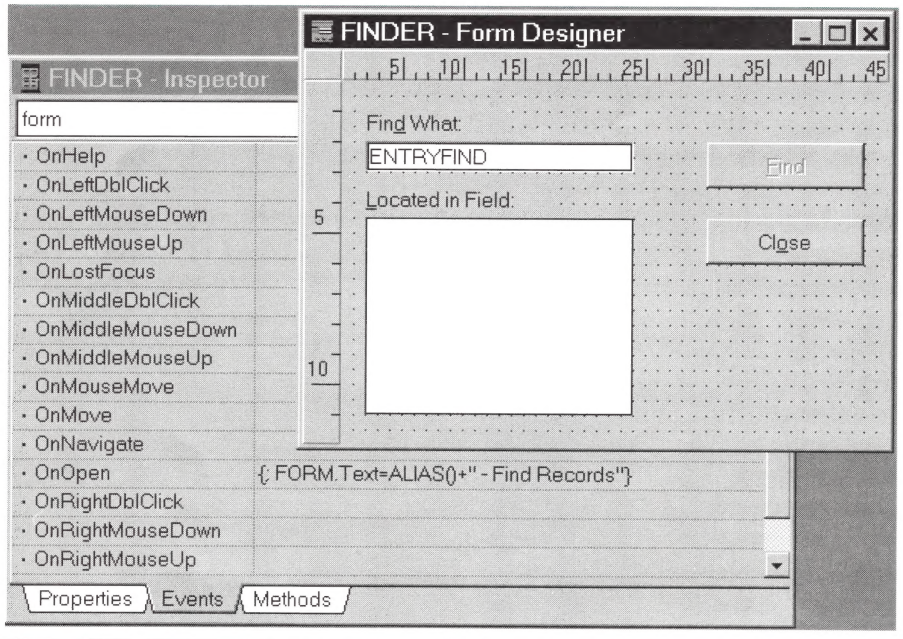
Adding events to the buttons

Buttons trigger actions, and you define the action in the OnClick event. The Close button requires a very simple action—one that closes the form. The Find action is a little more complicated and requires a procedure.

For the Close button, you can enter the following code block directly into the Inspector:

```
{ ; FORM.Close() }
```

Figure B



This is our custom Find Records dialog box.

Listing A: *The list box OnOpen event*

```
Procedure LISTFIELDS_OnOpen
  FORM.fields = NEW ARRAY()
  FOR i = 1 TO FLDCOUNT()
    IF TYPE(FIELD(i)) == "C" .OR. ;
       TYPE(FIELD(i)) == "M"
       FORM.fields.Add(FIELD(i))
    ENDIF
  NEXT
  THIS.DataSource = "ARRAY FORM.fields"
  THIS.SetFocus()
  * select first field
KEYBOARD "{DnArrow}" CLEAR
```

You can choose to leave off the braces that delimit the code block and let the Inspector insert them for you. However, you can't leave off the initial semicolon. If you do, Visual dBASE will execute your code block as a literal array, which will result in a data type mismatch.

The Find button needs to locate the first record that contains the value in the entry field. You can use macro substitution to dynamically build up and execute a LOCATE command by scoping the macro substitution variable as PRIVATE. Note that macro substitution doesn't work with variables that have the more insulated local scope.

Listing B shows the complete procedure for the Find button's OnClick event. Notice that the procedure converts to uppercase both the value to look for and the field to look in. This conversion makes the search case insensitive. The $ operator locates values anywhere in a field.

Listing B: *The Find button OnClick event*

```
Procedure PUSHFIND_OnClick
  PRIVATE macro
  macro= ["]+;
      UPPER(LTRIM(RTRIM( form.entryfind.value)));
      +[" $ UPPER(] + form.listfields.value + [)]
  LOCATE FOR &macro.
```

Finishing touches

Remember setting the enabled property of the Find button to false? Unless something sets it to true, this form isn't going to be of much use. Searches are valid only if someone enters a value into the entry field.

Several events fire when someone tries to commit changes to an entry field. Only one event fires when someone starts to modify the value in an entry field, and you refer to that action as the Key event. Here's the procedure to enable the Find button:

```
Procedure ENTRYFIND_Key(nChar)
  FORM.PushFind.enabled = .T.
RETURN nChar
```

Be sure to include the RETURN command. Without it, you won't be able to enter anything into the entry field.

You're almost done designing your custom Find dialog box. The only things left to do are to give the entry field a good name and to initialize the Value property.

Set the Name property to *EntryFind*. This name must match the name you use in the PUSHFIND_OnClick procedure.

The entry field needs to appear blank and allow as many spaces as you think someone will ever need to enter. As with the list box, you can initialize the value in the OnOpen event with a command in the form

```
{ ; THIS.VALUE = SPACE(50) }
```

Select the entry field's OnOpen event and enter this code block. Be sure to adjust the number of spaces as needed.

Using the custom Find dialog box

Your form is ready to go! Now you can attach it to a data entry form or a custom menu. In either case, you'll run the WFM from an OnClick event. We designed the dialog box as a modal operation that runs in the same session as the current form. It uses the active table to set itself up. Enter the code block

```
DO finder.wfm WITH .T.
```

to open the Find dialog box from an event.

Conclusion

While the native Find dialog box provides powerful search capabilities, it isn't available for data-entry forms that work with custom menus or that work outside the MDI frame window. In this article, we showed you how to create your own basic Find dialog box that works with character and memo fields. You can expand on this example to add support for numeric and date fields. You can also add other search options by dynamically adding options to the LOCATE command. ❖

Understanding SQL's SELECT

dBASE has supported Structured Query Language (SQL) in one form or another since dBASE IV 1.0. In the DOS version, there was a clear division between SQL code and native dBASE code. For instance, you had to issue the command *SET SQL ON* to invoke SQL mode. In that mode, dBASE imposed numerous limits on what you could do with the data. SQL program files were considered oddball enough to require their own file extension (PRS), and mixing native dBASE commands with SQL was an exercise reserved for the most adventurous among us.

Visual dBASE supports SQL without the separate SQL mode. Therefore, you can intermingle SQL commands with dBASE commands, and you no longer need to save SQL code in separate or uniquely named source files. In this article, we'll discuss when and why you should use SQL—if at all.

SQL SELECTively

The workhorse command in SQL is the SELECT statement. However, SQL's SELECT and dBASE's SELECT are very different commands. They perform distinct functions and their syntax is nothing alike, so don't confuse the two! SQL's SELECT command has roughly the same effect as issuing the USE and SET FILTER commands in dBASE.

To illustrate how SELECT and comparable dBASE commands work, we'll use the ORDERS and CUSTOMER tables from the Visual dBASE SAMPLES directory. Let's start simply by browsing a list of all customer names. In dBASE, you'd issue the commands

```
use customer
set fields to name
browse
```

To accomplish the same effect using SQL, you'd enter the commands

```
select name from customer
browse
```

Not much difference, is there? We saved a grand total of one line by using the SQL code.

Now let's look at a more complex query: Let's view all customer records by name, listing the customer number, name, and the date and amount of each sale for each customer. In dBASE, you'd use the following sequence of commands:

```
use customer order name in select()
use orders order customer_n in select()
select customer
set relation to customer_n into orders
set skip to orders
select customer
set fields to ;
customer->customer_n,customer->name, ;
orders->sale_date,orders->amt_paid
browse
```

In SQL, you'd enter the commands:

```
select c.customer_n, c.name, ;
       o.sale_date, o.amt_paid ;
       from "orders.dbf" o, "customer.dbf" c ;
       where o.customer_n = c.customer_n ;
       order by c.name
browse
```

Note that the SQL code is actually two lines long. We've folded the SELECT statement (divided it into several lines) for formatting purposes. As you can see, the SQL command is significantly more compact than the ten lines of dBASE code that produce the same results.

SQL SELECT statements can be very complex. They can include aggregate functions (SUM( ), MAX( ), and so on) for statistical analysis, a GROUP BY clause that many Visual dBASE users crave for querying purposes, and more. SQL SELECT can automatically save the selection set to a new table, and it automatically creates temporary indexes on an as-needed basis to satisfy ORDER BY clauses.

Using SQL SELECT statements with Visual dBASE forms is a little tricky. You can't specify a SELECT statement as the value for a form's View property. Instead, you create a QBE file that contains the SELECT statement, and then assign that QBE to the View property of the form. Visual dBASE's Query Designer doesn't work directly with SQL SELECT statements, so you'll have to create these statements manually.

No hard-and-fast rules govern the use of SQL in dBASE applications. However, if you're connected to SQL-based client/server databases that use SQL as a native language—such as Oracle, Informix, or Interbase—you'll often be better off using SQL.

But what about dBASE applications that use DBF or DB tables only? Judicious use of SELECT can make your code easier to read, and will even save you some time and debugging headaches. In most cases, SELECT won't make your programs any faster—but it can cut down the amount of code needed for a given application. ❖

Gearing up for the year 2000

by Joe Celko

If you've been watching your junk mail or reading the odd article in the newspapers or the computer trade press, you've heard about the year 2000, otherwise known as the "millennium problem." Such coverage generally consists of anecdotal horror stories about programs that used a two-digit year (year-in-century) format and blew up.

That sort of shallow story doesn't scare management and programming staff very much—we've seen programs crash before. However, the purpose of this article is to make you afraid. *Very* afraid.

Problems with the year 2000

There are three major problems with the date formatting of the year 2000 in computer systems. The year 2000

- has a lot of zeros in it
- is a leap year
- is a millennium year (Actually, the millennium doesn't change until 2001—but when 2000 clicks over, our computers will have a cataclysmic year.)

Let's consider each of these difficulties.

The zeros

I like to call the zero concern the "odometer problem" because it exists at the hardware, or system, level. If you're using a year-in-century format, the year 2000 will roll over like a car odometer that's reached its limit. Your application will then register the year as 1900, or something else other than 2000. (Paradox, for instance, reads the date "1/1/00" as January 1, 1900.)

To visualize the problem more clearly, set the date and time on your computer to just before midnight on December 31, 1999, and let it roll over. Don't look at the clock display—look at files created after midnight and see what date they bear. The most common dates are 1900, 1980, or 1984 (Macintoshes will reset to 1938).

This problem passes along to applications, but not always the way you'd expect. For instance, inputting the date 2000-01-01 into version 3.0 of Quicken on MS-DOS 6 works fine. But if you let the date wrap from 1999-12-31 into the year 2000, Quicken interprets the new date as 1901-01-01 rather than 2000.

The problem affects more than date fields. Timestamps are often buried inside encoding schemes. If an application uses the year-in-century format for the high-order digits of a serial numbering system, then any program that depends on increasing serial numbers will fail.

Hashing algorithms that use all or part of the system clock can blow up from a division by zero caused by the year 2000. Even if such algorithms don't crash, their statistical distribution may be changed drastically.

Sort packages, report writers, and other utilities use parameters that assume a year-in-century date format. You can't just make the date fields four digits long and set switches in one program. You need to list all the programs that access every data file on your PC.

Leap year

Each year has 365.2422 days, and every 400 years the fraction of the fractional day that was left over from the leap years accumulates. You've paid good money for programs that ignore this fact. Fortunately, Visual dBASE handles this portion of the year 2000 problem correctly. That is, if you add 1 to the date value 2/28/2000, Visual dBASE returns the date 2/29/2000, since 2000 is a leap year. However, if you simply enter *00* in a date field formatted as *MM/DD/YY*, Visual dBASE will interpret the year as 1900. To enter the year 2000 in Browse or Edit modes, you must first issue the SET CENTURY ON command, which displays dates in the format *MM/DD/YYYY*.

The millennium

The millennium problem was caused by programmers who didn't keep true dates in data fields—they tried to save space by using year-in-century formats. Some programs perform arithmetic and comparisons based on the year-in-century and not

on the year. Imagine you were born in 1955 and wanted to calculate the age you'd be in the year 2000. The difference between 55 and 00 is 55 years or -55 years—neither of which is close to the right answer (45 years old).

One horror story making the rounds deals with an inventory retention program that automatically issued orders to throw away good stock, thinking it was outdated. The inventory clerks working the warehouse floor caught the error and didn't throw away the merchandise. The accounting systems, on the other hand, didn't catch the error and mangled the books. The reordering system also failed to catch the error and cheerily kept reordering replacement merchandise, convinced that the original stock had spoiled on arrival.

Solutions

The first step toward dealing with these problems is to become aware of them. Second, subscribe to Internet newsgroups that deal with the topic. The two best are

- **LISTMANAGER@HOOKUP.NET**
  Send a message containing the text *SUBSCRIBE YEAR2000* to the above E-mail address to get on the list, which is run by Peter de Jager, an authority in the field.

- **Year 2000 Information Center home page**
  You can find this page on the World Wide Web at **http://www.year2000.com/cgi-bin/clock.cgi**.

Next, appoint a Year 2000 Task Group or Year 2000 Coordinator to inspect existing code and systems. The team can check and validate the existing data files for improper date fields.

You can take one of three approaches to dealing with the millennium problem: change the data, change the programs, or change both. If you make the changes without any trouble, you have to test them. Testing will cost you money, resources, and storage space. Does your company already have a testing method in place? testing teams? automated testing tools?

When you finish with these considerations, you have to change your paper forms, hard-copy reports, and screen displays. Look for displays with hard-coded century-in-title lines, report headers, and other displays. You don't want to title a report "date: _____19__" in the year 2000 just because you forgot to clean up a date field.

International standards

Until fairly recently, there's been little or no international agreement on the proper display format for dates. Americans put the month before the day, and the British do the reverse. When the date is 12/16/95 in Boston, it's 16/12/95 in London, 16.12.95 in Berlin, and 95-12-16 in Stockholm. Industry conventions are also inconsistent within countries.

You'll need to set up a program to convert your data so it conforms to ISO (International Organization for Standardization) 8601:1988, "Data Elements and Interchange Formats—Information Interchange—Representation of Dates and Times" as a corporate standard and EDIFACT for EDI messages. You can set the full ISO 8601 timestamp for either local time or UTC/GMT time. (UTC stands for Universal Coordinated Time, which replaced the older GMT, or Greenwich Mean Time.) For example, the timestamp string "1995-12-16T22:41:30+02" is a local time reference for the UTC timestamp "1995-12-16T22:41:30Z" if you're based in Stockholm, Sweden, which is two time zones west of longitude zero.

Standards

You won't find the text of ANSI and ISO standards on the Internet. However, both ANSI and ISO have online catalogs and other information you can access and search on the World Wide Web via their home pages:

ANSI **http://www.ansi.org**
ISO **http://www.iso.ch**

You can order both ANSI and ISO standards from ANSI by calling (212) 642-4900 between 8:45 a.m. and 4:45 p.m. EST. Check the ISO home page if you need to know who sells ISO standards in other countries. ❖

Making an alphabetical list easier to read

I maintain several databases and print alphabetical listings of my data for reference on a weekly basis. Do you know a way to divide those reports into new sections and start each section with the appropriate letter? For instance, I'd like to print *A* before names beginning with *A*, *B* before names beginning with *B*, and so on.

Brendan Schafer
Merrillville, Indiana

Crystal Reports for Visual dBASE makes it simple to create groups of related records and to label those groups. The easiest approach is to let the Expert help you create such a report. You just create a calculated field that returns the first initial of the last name. Then, you use that calculated field to define the groups within your report. Let's demonstrate how this works with an example.

If you're working from the Command window, issue the command *CREATE REPORT EXPERT PROMPT*. From the Navigator, select Reports and double-click the [Untitled] report icon.

When the Report Expert dialog box appears, click the Expert button. Then, choose any existing table. For this example, you might want to choose a table that contains customer names.

In the Expert's second step, the program asks you to choose a report type. Click the Next button to accept the default report type, Include Detail Records.

In the third step, select the fields you want to appear in the body of the report. As Figure A shows, you can choose one or more of the fields from the current table. In addition, you can click the Calculated Field... button to enter a custom expression.

Suppose you want to print your customer names using the format *Last, First*. To do so, click the Calculated Field... button. When the dialog box appears, type *fullname* in the Name field. Then press [Tab] to move to the Expression field, and enter the expression

```
TRIM(lname)+', '+fname
```

as shown in Figure B. (If you want the Expert to help you build an expression, just click the tool icon located at the right edge of the Expression field.)

Click OK, and the Expert will place the calculated field in the Selected column. At this point, you *could* click the Next button. However, in the next two steps, the Expert will ask you to choose the fields to determine record order and to define groups of records.

You're going to need a calculated field to tell the Expert to use the first initial of the last name as the group definition. Therefore, before proceeding to the next step, click the Calculated Field... button once more. This time, enter *initial* in the Name field and *LEFT(lname,1)* in the Expression field. This function call returns one character from the string in *lname*.

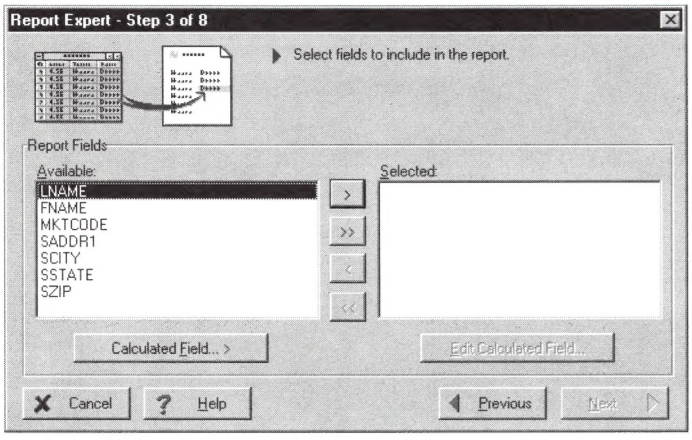
Click OK to save this calculated field definition. When you do, you'll notice that the expression appears with the *fullname* field in the Selected column. Don't worry—we'll delete that field from the body of the report later.

After you define the two calculated fields, click the Next button to complete the Expert's Step 3. When the Step 3 dialog box appears, double-click the expression that returns the full name. Then, click the Next button to complete Step 4.

The Step 4 dialog box asks you to choose a field (or expression) you want to use to group related records together visually. In the Available list, locate the custom expression for *initial* and double-click it. Click the Next button to complete Step 5. Since you probably don't need summary information in a report that doesn't include any numeric fields, click Next again to skip over Step 6.

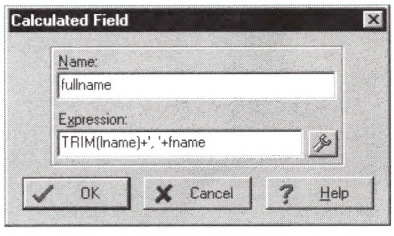
When the Step 7 dialog box appears, you can choose a layout (Tabular or Columnar) for the report. In addition, you can enter your own text

Figure A



You'll use this dialog box to tell Crystal Reports for dBASE what fields to include in the report.

Figure B



You define a calculated field like this one to return a customer's name in the form Last, First.

for the report title and toggle the Date and Page Number options. Then, click the Next button to continue.

In the Step 8 dialog box, click the Design Report button. When the Create Report dialog box appears, type *testrpt* and click OK. When you do, Crystal Reports for dBASE will open your report, which should look like the one shown in Figure C. As you can see, in its current form, the report includes the *initial* calculated field with the *fullname* field in the Details section of the report.

At this point, your report is almost ready. You just need to delete some extraneous information and move the *initial* calculated field to a new location in the report.

To begin, highlight the column label *fullname* in the Page header band, and press [Delete] to remove it. Do the same to remove the column label *initial*. In addition, you can delete the template for the numeric field (25,555) that appears in the Page footer band.

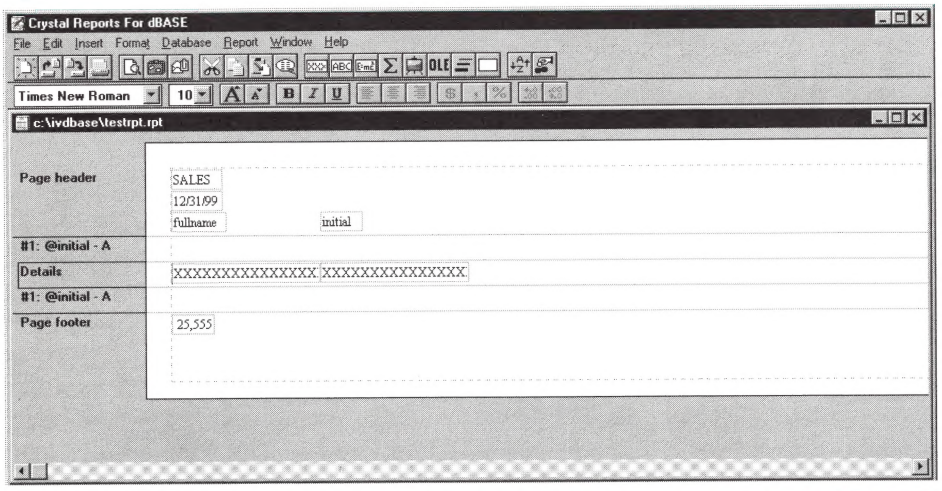
Next, select the second template in the Details band. Drag it up one level, and release the mouse to drop this calculated field into the group band, which is labeled #1.

Now, if you open the File menu, choose Print, and then choose Window, you'll see a report like the one shown in Figure D on the next page. As you can see, the *initial* calculated field returns the first initial of the last name in each group.

You can enhance your report even more by formatting the calculated field that returns the initial. First, close the Preview window. Then, with the report in Design mode, select the *initial* field in the group band. Open the Format menu and choose the Borders And Colors... option. Check the box for Border, select one of the border line options, and click OK.

After Crystal Reports applies the border, the field remains selected. Open the

Figure C



After completing the Expert design process, your report form should look like this one.

INSIDE VISUAL dBASE™

Inside Visual dBASE (ISSN 1084-1970) is published monthly by The Cobb Group.

Staff:

Contributing Editor-in-Chief ..... Keith G. Chuvala
Associate Editors-in-Chief ..... Jeff E. Davis
Tiffany M. Taylor
Publications Coordinator ..... Maureen Spencer
Editors ..... Laura Merrill
Joan McKim
Elisabeth Pehlke
Production Artist ..... Alison Schwarz
Product Group Manager ..... Mike Stephens
Circulation Manager ..... Mike Schroeder
Associate Publisher ..... Mark Kimbell
VP/Publisher ..... Mark Crane
President/CEO ..... J. Thomas Cottingham

Address:

Please send tips, special requests, and other correspondence to

The Editor, *Inside Visual dBASE*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

E-mail address: visual_dbase_win@merlin.cobb.zd.com

For subscriptions, fulfillment questions, and requests for bulk orders, address your letters to

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

E-mail address: customer_relations@merlin.cobb.zd.com

Phone:

Toll free US ..... (800) 223-8720
Toll free UK ..... (0800) 961897
Local ..... (502) 493-3300
Customer Relations Fax ..... (502) 491-8050
Editorial Department Fax ..... (502) 491-3433

Back Issues:

To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $8.50 each, $8.95 outside the US. You can pay with MasterCard, VISA, Discover, or American Express, or we can bill you.

Postmaster:

Second Class Postage Paid in Louisville, KY.

Postmaster: Send address changes to:

Inside Visual dBASE
P.O. Box 35160
Louisville, KY 40232

Advertising:

For information about advertising in Cobb Group journals, contact Tracee Bell Troutt at (800) 223-8720, ext. 430.

Bulk Sales:

For information about bulk/group subscription sales, please contact Customer Relations at (800) 223-8720.

Prices:

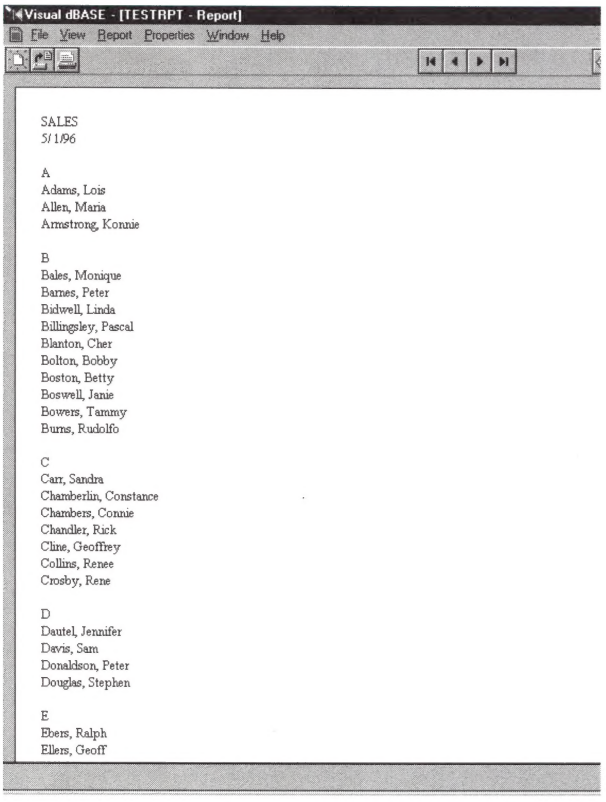
Domestic ..... $79/yr ($8.50 each)
Outside US ..... $89/yr ($8.95 each)

Format... menu again and choose Font. When the Font dialog box appears, choose the 14-point size, Bold, and click OK.

Now, open the File menu, choose Print..., and then choose Window. This time, the group labels appear in the 14-point bold font. However, as Figure E shows, the border itself extends well beyond the initial!

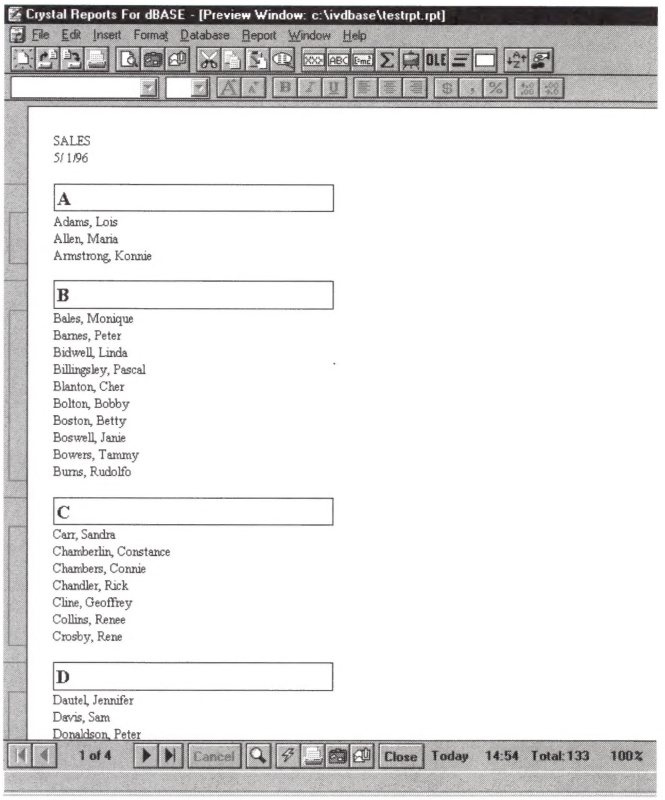
Fortunately, you can adjust the border's width without leaving the Preview Window. First, click on the border for the first group—the one around the letter *A*. Then, drag the right side of the border to the left until the spacing looks right. As you adjust the spacing for the first *initial* field, the program automatically adjusts that field in each group. Figure F, a printout of the first page of our sample report, illustrates how much a formatted group label can improve the readability of your reports. ❖

Figure D



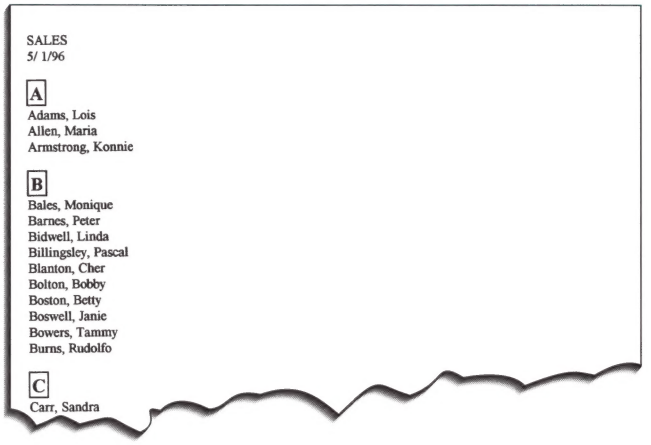
Grouping records by the first initial makes it easy to find a name in an alphabetical list.

Figure E



You can fine-tune this report's appearance by adjusting the width of the border around the initials.

Figure F



Adjusting the border width around the initials greatly improves your report's readability.